中編

システムの記事

FlexRayの実現に向けた時間駆動型通信リアルタイムOSを開発

――TT-OS仕様におけるトリガ・タイミングとクロック同期

服部博行

ここでは，タイム・トリガ（時間駆動）方式の通信制御を実現するために開発されたリアルタイムOS「TT-OS」の仕様を解説する．FlexRayネットワークではあらかじめ定められたスケジュールに従って通信が行われる．そのため，ネットワーク全体の時間とそれぞれのノード（ECU；電子制御ユニット）が持つ時間を合わせる（同期をとる）必要がある．正常通信時，TT-OSではFlexRay通信コントローラのタイマのみでタイム・トリガ処理を行っているため，時間補正が不要となる．

（編集部）

前編（本誌2006年4月号，pp.115-121）では，FlexRay通信を実現するためのリアルタイムOSには，タイム・トリガ機能の提供が必要であることを述べました．また，この要求に合わせて筆者らが新しくタイム・トリガOS（TT-OS）を開発した経緯についても説明しました．TT-OSは，OSEK/VDX OS（以下，OSEK OS）注1にタイム・トリガ機能を提供するタイム・トリガ・モジュール（TTM）を追加して実現します．すなわち，TT-OS＝OSEK OS＋TTMという構成になっています（**図1**）．

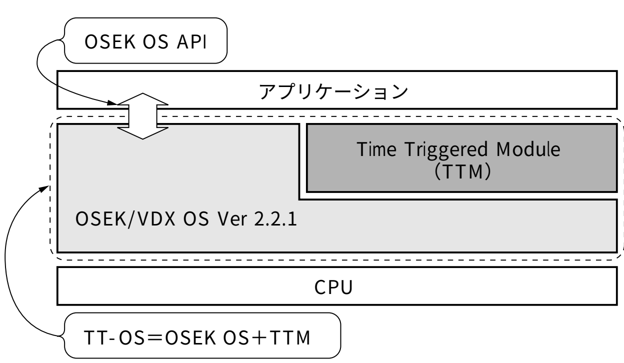


図1　TT-OSの構成
TT-OSはOSEK OS＋TTMで構成される．

1　TT-OSの特徴 ―― TTMの役割とトリガ取得の方法

TT-OSはOSEK/VDXがタイム・トリガ方式の通信を実現するために仕様策定したOSEK/VDX Time-Triggered Operating System（以下OSEKtime）(3)を参考にしていますが，動作性能上の仕様は大きく異なっています（なぜ違う仕様になったかは前編を参照）．またTT-OSは，以下に示すようにメモリ・リソースを抑え，アプリケーションの移植性なども考慮した構成となっています．

1）メモリ・リソース

前述したとおり，TT-OSはOSEK OSにTTMを組み合わせて実現します．そのため，タイム・トリガ機能を実現するにはTTMを追加するだけです．

一方，TT-OS以外のOSの多くは，タイム・トリガ機能を提供するOSとイベント・ドリブン機能を提供するOSを利用した，いわゆるタンデム型を採用しています．複数のOSを実装することになるので，それだけメモリなどのリソースも必要となります（**表1**）．

注1：OSEK/VDXは，ECU間のインターフェースに関する仕様．OS，通信，ネットワーク管理について定義している．本仕様を定めている欧州の自動車メーカを中心とした仕様策定団体の名称でもある．

KeyWord　FlexRay，ECU，TT-OS，OSEK/VDX OS，OSEKtime，TTM，OIL，イベント・ドリブン，タイム・トリガ，同期

2）アプリケーションの移植性

TT-OSを用いた場合，既存のアプリケーションで使用されているOSEK OS サービスやイベント・ドリブンによる制御方式を利用できます．そのため，FlexRay通信を利用したい場合でも，既存のアプリケーションを大幅に変更することなく実現できます．

一方，ほかのタイム・トリガOSを利用する場合，その考えかたの違いや利用するAPI（application programming interface）をすべてアプリケーションで吸収する必要があります．利用するAPIなどの変更は，ある程度単純作業で置き換えることができるかもしれませんが，イベント・ドリブンからタイム・トリガへの制御方式の変更は，かなりたいへんな作業となるでしょう．これを，エンジン制御を例にとって考えてみます．現在のエンジン制御は回転軸に同期した制御を基本としていますが，タイム・トリガ方式のOSでは回転軸に同期したイベントを発生できません．すると，制御方式から見直さなければならなくなります．このように，制御方式の異なるOSに変更するには，従来の制御方法を見直したり，時間的制約が守れるかどうかを検証するなど，多くの作業が必要となり，事実上，移植は難しいといわざるをえません．

また，移植性という意味ではありませんが，OSEKtimeとOSEK OSを用いる構成を選択した場合，アプリケーションからは二つの異なるOSにサービスを要求することとなります．つまり，アプリケーション開発者は双方のOSの知識を持つ必要があります．

● TTMの役割はタスクの起動をOSに要求すること

TT-OSのタスクへのサービスは，OSEK OSから提供します．では，TTMはいったい何を行うミドルウェアなのでしょうか？

TTMは，静的に決定されたスケジュール・テーブルに従い，OSEK OSに処理を要求します．TT-OSでは，このスケジュール・テーブルをOIL（OSEK Implementation Language）で記述します（このOILは，OSEK/VDX が規定したOIL仕様Version 2.5[(7)]をベースに，筆者らが独自に拡張したフォーマットを規定した）．OILで記述されたスケジュール・テーブルは，システム・ジェネレータ（SG）を用いて二つのスケジュール・テーブルに分割されます．その後，コンパイラなどによりバイナリに変換され，それをTTMが利用します．TTM は，OSEK OSのカテゴリ2の割り込み（ISR2）[注2]によって起動します．そして，割り込みタイミングに対応した処理をスケジュール・テーブルから検索し，OSEK OSに処理要求を出します．同時に，次に動作するべき時間を検索してTTMの起動を設定します．

表1　TT-OSに必要なメモリ・リソース

	ROM		RAM
	コード	データ	データ
TOPPERS/OSEKカーネル	5,851	64	103
TTM	1,804	790	13

※単位はバイト
※TT-OSはTOPPERS/OSEK＋TTMで算出できる
※計測はルネサス テクノロジ製コンパイラを利用した．ターゲットCPUはM32C/85

TTMからOSEK OSに要求する動作はタスクの起動であり，TTMが直接タイム・トリガ・タスクを動作させるわけではありません．つまり，TTMは要求の発行を行うだけです．起動要求が行われたタスクは，ほかのタスクと同じようにタスクに割り当てられた優先順位に従ってOSEK OSにより起動順序が決定され，サービスが提供されます．

● 指定時間内に処理が終わっているかどうかを確認する

TTMはタイム・トリガ・タスクの起動をOSへ要求するほか，あらかじめ登録されたコールバック関数の呼び出しを行うこともできます．これは，OSEK OSにタスクの起動要求を行ってから起動順序が決定されるまで待つことが難しい処理を，TTMから直接実行する機能です．そのため，OSEKtimeと同等の処理応答を実現できます．ただし，この機能は割り込みハンドラ内で実行されるため，長時間かかる処理を行ったり，多用することは運用上禁止する必要があります．

さらにタイム・トリガを実現するための重要な機能として，デッドライン監視があります．タイム・トリガ方式の通信を行う場合，ある時間から処理を開始してある時間までに処理が完了していないと，システム全体で不整合が発生する可能性があります．このため，指定時間までに処理（例えば通信データの作成など）が完了しているかどうかを確認する機能がデッドライン監視です．この機能は，なんらかの原因により処理が完了していないときは，これを正

注2：OSEK OSの割り込みサービス・ルーチン（ISR：Interrupt Service Routine）は二つのカテゴリ（カテゴリ1，カテゴリ2）に分かれる．カテゴリ1の割り込みサービス・ルーチンはOSから独立しており，OSEK OSのAPIを呼び出せないが，カテゴリ2では自由に呼び出すことができる．

確に把握して必要とする処理を実行します．

このように，TT-OSはタイム・トリガ方式の処理とイベント・ドリブン方式の処理を両立しつつ，双方のサービスに必要な要件を正確に分析し，それぞれのサービスに必要な機能に対応しています．

● ネットワーク時間を起動トリガとして使用する

TTMの動作タイミングをつかさどる基本時間は，外部からの割り込みにより取得します．TT-OSはFlexRay通信機能を実現するために開発されており，基本時間を得るための割り込みはFlexRay通信コントローラのタイマを利用します．

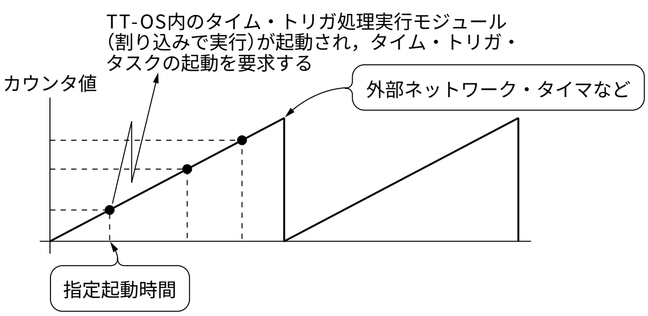


図2　外部タイマとタイム・トリガ・サービスの関係

TT-OSでは，カウンタとしてFlexRay通信コントローラのタイマを利用している．FlexRay通信における指定サイクルの指定時間にタイマ割り込みを設定することにより，プロセッサに割り込みが通知される．プロセッサではFlexRay通信コントローラからの割り込みをTTM処理のトリガ条件としているため，TTMではそのタイミングで必要となるタイム・トリガ・サービスを実行する．

タイム・トリガ型OSの多くは，外部時間[注3]と内部時間[注4]を別々に管理し，この二つの時間の差を周期的に計算し，周期開始時のオフセット処理やクロックの伸張などを用いてそれぞれの時間を補正します．このような方法では，周期タイミングごとにオフセット計算や比例計算を行わなくてはなりません．そのため，周期タイミングにおける処理負荷や周期内の起動誤差（周期のはじめと終わりなど）が大きくなることが考えられます．

これらの問題点を解決するため，TT-OSでは指定サイクル[注5]の指定時間にFlexRay通信コントローラに割り込みを発生させ，これをTTMに通達する方式をとっています（**図2**）．この方式では，タイム・トリガ・タスクの起動トリガとなる割り込みに外部時間をそのまま利用するため，正確なタイミングをプロセッサに伝達することができます．また，周期ごとの補正処理が不要となります．

注3：FlexRayの場合，外部時間はノード（ECU）外部のネットワーク全体に共通した時間を示し，「グローバル時間」と呼ばれることが多い．
注4：FlexRayの場合，ECU内部の時間を示し，「ローカル時間」と呼ばれる場合が多い．
注5：FlexRay通信におけるデータ送信の単位を「コミュニケーション・サイクル」という．ここでは単に「サイクル」としている．

Column 01　CAN通信はイベント・ドリブン方式だけで利用されるの？

その答えは“No”です．

CANでは，その拡張仕様としてTT CAN（time-triggered communication on CAN）が定義されています．このTT CANは，CANの物理層をそのまま利用し，CANの上位層にタイム・トリガ処理を追加することで実現しています．そのため，従来のCANの特徴をそのまま継承しつつ，タイム・トリガの特性を取り入れています．具体的には，CANの特定メッセージに時間情報を設定し，各ノードに配信します．それぞれのノードはその時間情報をネットワーク時間と判断し，ネットワーク全体で同一の時間を利用してタイム・トリガを実現する方式のようです．仕様上では，イベント・ドリブン通信とタイム・トリガ通信の両方に対応できるものと考えられます．

このようにCAN通信でもタイム・トリガ方式の利用は可能ですが，タイム・トリガ方式は既存のCAN通信への追加仕様であるため，タイム・トリガの特徴がすべて網羅されているかどうかは定かではありません．

また，同種の方式としてTTP/C（time triggered protocol/Class C）があります．これは，SAEのClass Cに適応した仕様です（**表A-1**）．この仕様は，安全系ネットワークに適用する目的で策定された仕様であり，FlexRay通信が目標としているX-by-wireを見据えた仕様となっています．この仕様でもCANの物理層を利用できます．

TTP/Cは，FlexRay通信が策定される前は次世代のタイム・トリガ型通信の最有力候補でした．しかし，2003年にTTP/Cの規格を策定する主要メンバがFlexRayコンソーシアムに加盟したことにより，X-by-wireの実現を目ざした通信プロトコルとしてFlexRayが有力な候補になりました．

表A-1　車載LANのデータ通信速度

SAEのクラス	速　度	用　途
A	最大10kbps（ボディ系）	ランプ，ライト，ドアロックなど
B	10kbps～125kbps（ステータス情報系）	電子メータ，故障診断など
C	125kbps～1Mbps（リアルタイム制御系）	エンジン，トランスミッション，ブレーキなど
D	5Mbps以上（マルチメディア系）	カー・ナビゲーション・システム，オーディオなど

反面，FlexRay通信コントローラの機能を利用するため，デバイス依存の部分が問題になります（下掲のコラム「OSEKtimeと相性の良いプロセッサ」を参照）．この問題を解決するため，筆者らは国内半導体メーカのFlexRay通信コントローラについて調査しました．その結果，これらの半導体メーカはいずれも同一ベンダのFlexRay通信コントローラIPコアを利用して製品を開発すると発表しており，とくに国内での利用は問題にならないと判断しました．別のIPコアを採用した通信コントローラを用いた場合でも，FlexRayデバイス・ドライバで対応することにより，TTMへの影響を最小限に抑えます．

● 通信エラー時でもなんらかのタイミングで処理を行う

TT-OSで基本時間を取得するために用いているFlexRay通信コントローラのタイマは，FlexRay通信が正常な場合にのみ動作します．したがって，FlexRay通信が正常に通信を行っている場合は前述したとおり，所望とするタイミングを正確に取得できます．

しかし，なんらかの理由（スタートアップ時や障害発生時など）により通信エラー状態となると，外部時間が得られなくなります．このような場合を想定し，TTM上ではつねに内部時間作成用タイマを起動させます．具体的には，プロセッサ内部のインテリジェント・タイマ（タイマ値'0'からカウントアップし，指定したタイマ値になると割り込みを発生させ，タイマ値を初期化して再スタートするタイマ）を用いて内部カウンタを動作させます．このとき，タイマ満了値を外部時間周期より長く設定し（デフォルトでは外部時間周期の110％），正常時はFlexRay通信コントローラからのサイクル開始割り込みによってタイマ値を初期化します．この初期化が内部時間の補正（いわゆるタイマのリセット）となるので，サイクルごとの複雑な演算を必要としません．また，かりに同期ずれなどの通信障害が発生しても，次のような場合は即座に利用するタイマをプロセッサ内部のタイマに切り替えます．

- FlexRay通信コントローラから通知されるエラー割り込みを捕捉した場合

Column 02　OSEKtimeと相性の良いプロセッサ

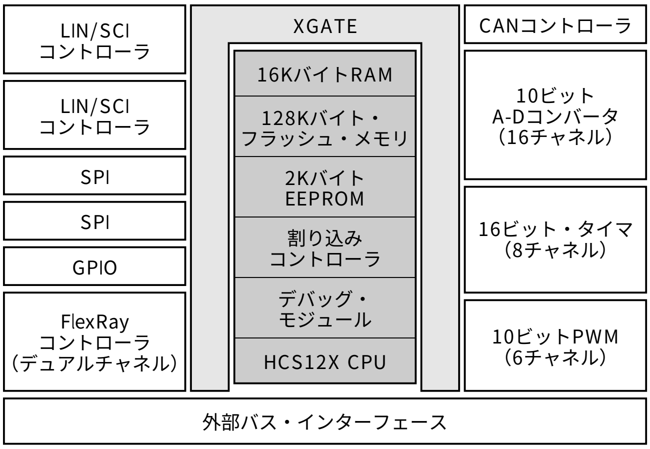
先日，筆者は米国Freescale Semiconductor社のFlexRay対応16ビット・マイコン「MC9S12XFR128」の構成を見る機会がありました．このマイコンは，CPUコア「HCS12X」のほか，「XGATE」と呼ばれるデバイス処理用コアが搭載されており，一種のマルチプロセッサ構成をとっています（**図B-1**）．XGATEは，多くのプロセッサが不得意とする，ビットのシフト命令やビット演算を高速に処理する機能を備えており，通信時のデータ処理を高速に実行できるようにくふうしてあります．

一見すると，XGATEは周辺回路制御用のコプロセッサのような存在に見えますが，ちょっと考えかたを変えてみると，このマイコンを利用してOSEKtimeの問題点を解決できるかもしれません．つまり，CPUコアでOSEK OSを動作させ，XGATEでOSEKtimeを動作させるのです．そうすれば，それぞれのコアは並列に動作するため，OSEK OSのタスクはOSEKtimeのアイドル時間を待つことなく実行できます．

同社がOSEKtimeの問題点を解消するために本マイコンを開発していたとしたら，OSEKtimeの特徴をかなり細かく検討していたのだろうと想像できます．同社の日本法人の方と話す機会があったので問い合わせてみましたが，実際にはそのような目的でマルチコアになったのではない，とのことでした．

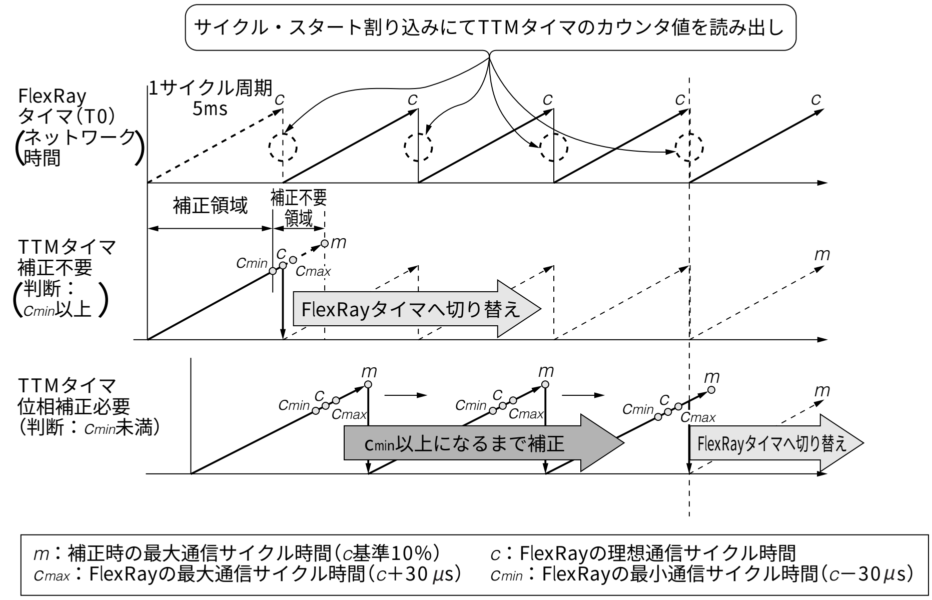
すり合わせの開発により，いろいろな処理を同じマイコンで実行させてしまう日本的な発想を引き継いだTT-OSは，やはり日本人が考え出した手法であると感じてしまいます．一方，それぞれ異なる機能をそれぞれのコアで処理させる方式は「組み合わせ型」って言えませんか？ だとしたら，欧米的な方法なのでしょうか？

いずれにしても，OSEKtimeの問題点をハードウェアで克服するのに，マルチプロセッサという方法は優れていると思います．

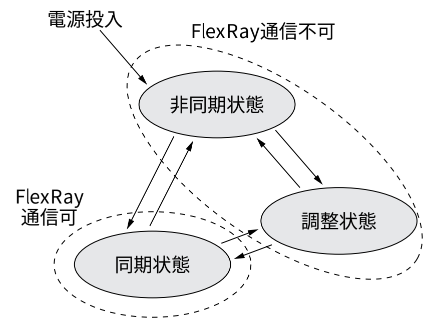


CAN：controller area network　LIN：local interconnect interface
SCI：serial communication interface　SPI：serial peripheral interface

図B-1　Freescale社のFlexRay通信コントローラ搭載プロセッサ(8)
HCS12XでOSEK OSを，XGATEでOSEKtimeを動かす，という構成はどうだろう．



(a) タイマ切り替え



(b) TTMの状態モデル

図3　TTM のタイマ切り替え制御

なんらかの通信障害などにより同期ずれが発生したときの，その復旧過程を示す．FlexRay 通信コントローラではほかのノードとの同期が確立し，デバイスからサイクル開始割り込みの通達がくると，復旧した状態になる(TT-OS はローカル時間とグローバル時間が一致していない)．ローカル・タイマの周期はネットワーク・サイクル周期より長いため，ローカル・タイマの満了によるタイマ・リセットでネットワーク時間との差が徐々に縮まる．サイクル開始割り込みが発生したとき，ローカル・タイマのカウンタ値が補正処理を必要としないタイミングであれば，ローカル・タイマをリセットし，FlexRay タイマへ切り替えて TT-OS は正常通信できる状態に移る．

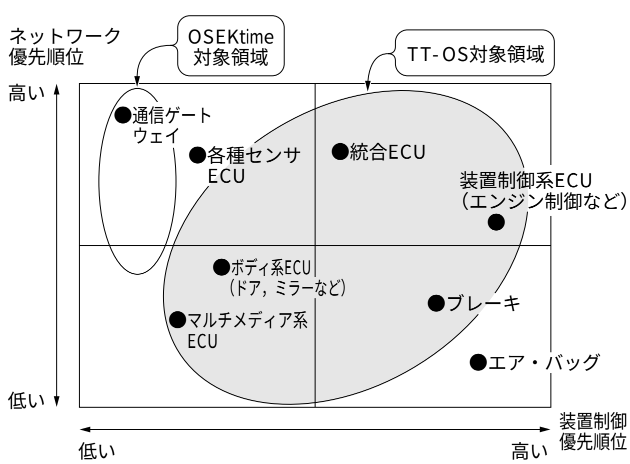


図4　TT-OS とOSEKtime の対象領域

通信がおもな処理となるゲートウェイなどでは，OSEKtime は非常に利用しやすく，矛盾のないシステム構築が可能となる．それ以外では，タイム・トリガ処理が無条件に高い優先順位となるので，システム構築上かなりの制限を受ける．

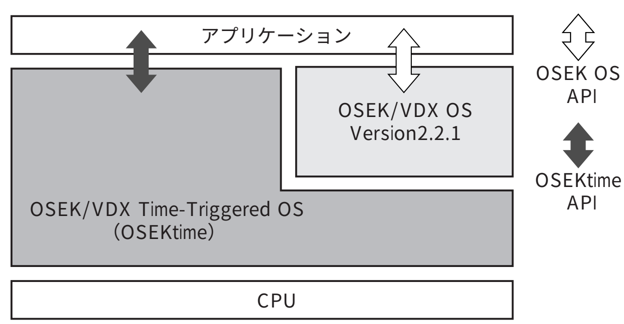


図5　OSEKtime のソフトウェア構成図

OSEK OS はOSEKtime のサブシステムに相当する．この構成では，OSEK OS と比べて OSEKtime は絶対的な権限を持っている．

- 通信コントローラから発行されるはずのサイクル開始割り込みが通達される前に，インテリジェント・タイマの満了割り込みを確認した場合

このように同期通信時ほど正確なタイミングではないにしても，通信エラー状態においてタイム・トリガのサービスを提供し，処理を継続させます(通信が行われないため，正確なタイミングは不要だと考える)．

● 内部と外部の周期の差を作って同期をとる

ノード間の同期が完了すると，FlexRay 通信コントローラでは正常時と同じようにサイクルの開始割り込みが発生します．この割り込みのタイミングとそのノードの内部時間(インテリジェント・タイマのカウントアップ時間)を比較し，補正が不要であると判断した時点でインテリジェン

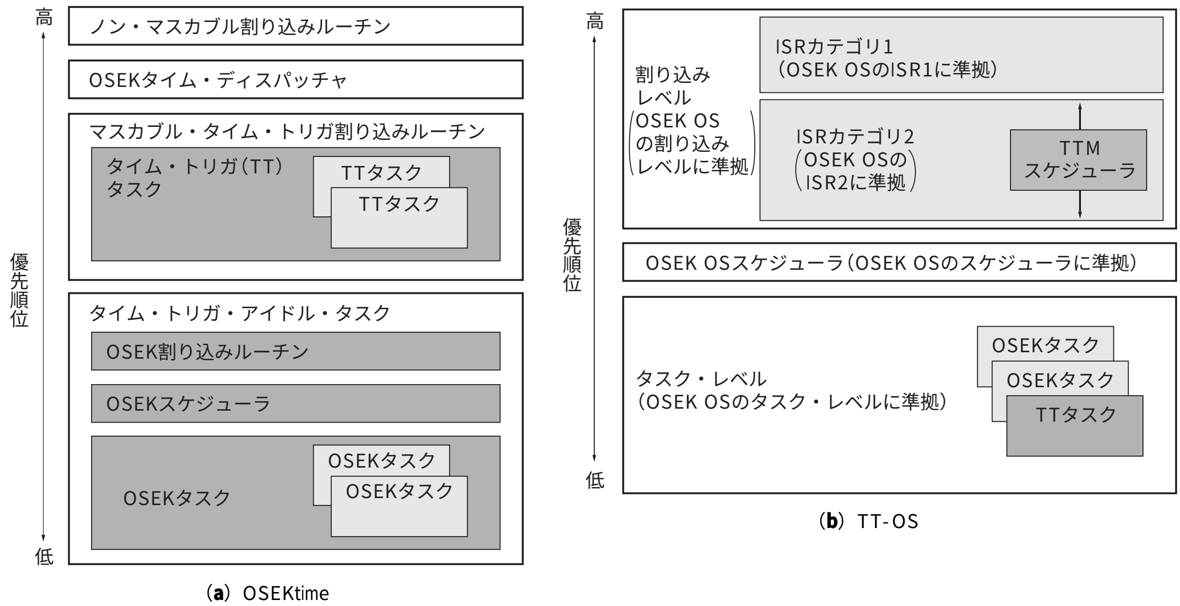


図6 プロセッシング・レベル
(a)のように，OSEKtimeでは非タイム・トリガ(イベント・ドリブン)の制御処理はつねにタイム・トリガで実行される処理よりも後回しにされる．一方，TT-OSでは(b)のようにイベント・ドリブン型かタイム・トリガ型かにかかわらず，タスクの起動順序はOSEK OSの与える優先順位で決まる．

ト・タイマを初期化し，クロック同期が完了します．

ここで重要となる点は，外部時間の周期と内部時間の周期が同じである場合，両者の周期開始タイミングの差が縮まらないということです．そこで，前述したようにインテリジェント・タイマを外部時間周期より長めに設定することで，わざと外部時間周期と内部時間周期の差を作ります．この差を利用してクロック同期を行います．また同期完了は，補正不要領域にあるかどうかで判断できます(**図3**)．

FlexRayでは，何回データが送信されたか(つまり，何回サイクルが繰り返されたか)を数えています．0から始まり63になるとカウンタは再び0に戻り，カウントアップします．サイクル開始のタイミングが内部時間と外部時間で一致したとしても，ある時点で実行しているサイクル(のカウント値)が一致していないと問題が起こります．このサイクルの同期にも前述した周期時間差を利用しています．これにより，サイクルの抜けやタイム・トリガ処理の抜けを防止することができます．

2 TT-OS vs. OSEKtime

次に，TT-OSのベースとなったOSEKtimeとの相違点をまとめます．TT-OSは既存のOSEK OSと組み合わせてタイム・トリガ方式の通信に必要なサービスを提供することを前提としています．TT-OSとOSEKtimeが対象とする領域はおおよそ**図4**のようになります．

● タイム・トリガを優遇しないソフト構成で設計を容易に

OSEKtimeのソフトウェアの構成を**図5**に示します(TT-OSのソフトウェア構成は**図1**を参照)．この構成により，OSEKtimeでは以下のような制限事項があります．

- OSEK OSおよびOSEK OS環境下で動作するアプリケーションは，OSEKtimeの空き時間(アイドル時間)にのみ実行される
- OSEKtimeのほうが，OSEK OSやOSEK OSのサブシステムよりも優先度が高い
- OSEK OSのノンプリエンプティブ・タスク(一つのタスクを処理中はほかの処理を行わない)であってもOSEKtime環境下のタイム・トリガ・タスクにはプリエンプト(実行タスクが中断)される．

これらの制限は，システム全体でタイム・トリガによる制御しか動作の保証が得られない，つまりOSEK OS上で重要な制御を実行できないことを意味しています．このことは，OSEK OS上で制御されるアプリケーションの設計を難しくしています．

一方，TT-OSではこうしたOSEKtimeの制限事項を排除するため，OSEK OSの下でタイム・トリガ方式の処理

表2
API比較表
太字で示しているものが，TT-OSに新しく追加されたAPI．

機　能	TT-OS	OSEKtime
タスクIDの取得	OSEK OSのAPIを利用（GetTaskID）	ttGetTaskID
タスク状態の取得	OSEK OSのAPIを利用（GetTaskState）	ttGetTaskStatus
アプリケーション・モードの変更	―	ttSwitchAppMode
現在のアプリケーション・モードの取得	OSEK OSのAPIを利用（GetActiveApplicationMode）	ttGetActiveApplicationMode
OSEK OSの状態の取得	―	ttGetOSEKOSState
指定アプリケーション・モードでOSを始動する	OSEK OSのAPIを利用（StartOS）	ttStartOS
OSを停止させる	OSEK OSのAPIを利用（ShutdownOS）	ttShutdownOS
エラー発生時のフック・ルーチン	**ttmErrorHook**	ttErrorHook
スタートアップ時のフック・ルーチン	**ttmStartupHook**	ttStartupHook
OS停止前のフック・ルーチン	OSEK OSのAPIを利用（ShutdownHook）	ttShutdownHook
同期から外れたときのフック・ルーチン	**ttmNetworkNoSyncHook**	―
同期に入ったときのフック・ルーチン	**ttmNetworkSyncHook**	―
外部時間との同期状態を得る	**ttmGetSyncStatus**	ttGetOSSyncStatus
外部時間を得る	**ttmGetNetworkTime**	ttSyncTimes
実行中のOSがサービスを提供しているサイクルを得る	**ttmGetActionCycleCount**	―
現在利用中の時間を得る	**ttmGetTime**	ttSyncTimes

を管理することにしました．これにより，タイム・トリガやイベント・ドリブンといった処理方式にかかわらず，システム全体で優先順位を指定することができます．タイム・トリガ・タスクの起動要求タイミングはOSEKtimeと類似するものの，起動要求後の起動タイミングはOSEK OSの優先順位により決定されます．そのためアプリケーションの設計時に動作を検証しやすくなり，かつ従来のアプリケーションの移植が容易となります．

● プロセッシング・レベルはOSEK OSを踏襲

OSEKtimeのプロセッシング・レベルは，タイム・トリガ処理が優先され，OSEK OSが管理するすべてのタスクはタイム・トリガ処理の空き時間に実行されます（**図6(a)**）．タイム・トリガで実行される制御は，あらかじめ静的に定められた時間でサービスが提供されるため，時間を基準にしたアプリケーションの設計は容易ですが，時間でないもの（イベント・ドリブン）を基準にしている場合は設計が非常に難しくなります．

また，OSEKtimeは異なる2種類のOS（OSEKtime，OSEK OS）を搭載しており，近年多く見られるデュアルOSやタンデムOSと類似しています．このような構成をとった場合，スケジューラが多段構成となり，上段に配置されるタスク（この場合はOSEK OS管理タスク）の起動はオーバヘッドが大きくなります．

TT-OSの場合，タイム・トリガ方式のタスクの起動はTTMスケジューラから要求されます（**図6(b)**）．タスク起動要求はOSEK OSのシステム・コールを利用し，タスク起動処理はOSEK OSのスケジューラが行います．すなわちタイム・トリガ方式のタスクであっても，OSEK OSが管理するイベント・ドリブン方式のタスクと同じように管理されます．これは，タイム・トリガ方式のタスクとイベント・ドリブン方式のタスクでは起動要求の方法が異なるだけで，タスクの起動順序はOSEK OSの管理の下で平等に行われることを意味します．

このほか，上述したようにTT-OSは通常の通信状態で時間補正処理が不要となるなど，ノードの同期についてもOSEKtimeと比べて優位性があります．また，TT-OSはOSEK OSにTTMというモジュールを追加することで実現しているため，追加のAPIは少なく，OSEKtimeで提供されている多くのAPIと同等の機能がOSEK OSのAPIを利用しています．そのため，OSEK OSとの親和性が高く，タイム・トリガ処理を実現するために追加するオブジェクト・サイズは小さいという特徴があります（**表2**，**表3**）．

次回は，TT-OSと同時に開発したFlexRay通信ミドルウェアについて解説します．

表3　OSEKtimeとTT-OSの性能比較

機　能	TT-OS	OSEKtime
タスクのブロッキング（外部イベント待ち）	ブロッキング可．適合クラスにより異なる（高優先度割り込み，タスク，待ちイベント）	ブロッキング不可．起動要求後，直ちに実行
タスク起動時刻やスケジュールは静的テーブルに記述	OILにてスケジュール・テーブルを記述し，静的なテーブルを利用してタイム・トリガ処理を行う	仕様上，なんらかの方法で静的テーブルを作成し，利用する
デッドライン監視	サポートあり．OSEKtimeの仕様と同等	サポートあり
タイム・トリガ・タイミングのコールバック	TTMからタイム・トリガ・タイミングでコールバック関数の呼び出しが可能	仕様上，サポートなし
タスクのプリエンプト	適合クラスにより異なる	タイム・トリガ・タスクはつねにほかのタイム・トリガ・タスクをプリエンプトする．また，すべてのOSEKタスクをプリエンプトする（プリエンプト禁止のタスクも含む）
タスク起動	OSEK OSスケジューラ	OSEKtimeディスパッチャ
タイム・トリガ処理部位のプロセッシング・レベル	OSEK OS ISR2	NMIを除いた最高レベル割り込み
タイム・トリガ静的タスク優先度	タイム・トリガ・タスクは静的タスク優先度を持つ	タイム・トリガ・タスクは静的タスク優先度を持たない
スケジューリング名称	優先度ベース・スケジューリング（優先度上限プロトコル）	スタック・ベース・スケジューリング
スケジューリング・ポリシ	ワースト・ケースの実行時間によるタスク設計は不可（タスク・ブロッキングがあるため）	ワースト・ケースの実行時間による設計が可能
非タイム・トリガ・タスクの扱い	タイム・トリガ・タスクと非タイム・トリガ・タスクの差はない．すべてOSEK OSの管理下	OSEKtime管理下にある処理がまったく行われていないアイドル時間に非タイム・トリガ・タスクは実行される．OSEK OS管理下の割り込みも含む
アプリケーション・モード	OSEK OS仕様を継承	ディスパッチャ・テーブルと連動したアプリケーション・モード．動作中の変更が可能
割り込みポリシ	OSEK OS 仕様を継承	OSEKtime独自ポリシ．OSEKtime処理が最優先．同一割り込みが短時間に入らないように，割り込みごとに禁止処理がある．また，割り込み許可をディスパッチャ・テーブルで静的に記述する（割り込みスケジュールがほぼ静的に決定されていることが条件となっているように思われる）
割り込み禁止許可	OS，アプリケーションともに可能	アプリケーションの操作を制限
外部時間同期処理	同期時：不要（外部時間を利用） 非同期時：ローカル・タイマのリセット処理	サイクルごと，もしくはラウンド（64サイクル）で演算子補正する
タイマの切り替え処理	必要．同期時と非同期時により利用するタイマが異なる	不要
同期方法	2種類の同期方法を提供	3種類の同期方法を提供

参考・引用*文献

(1) Howard Curtis and Robert France ; Time Triggered Protocol (TTP/C) : A safety-Critical System Protocol, *EE382C Literature Survey*, Oct 1999.
(2) CIA Webサイト，http://www.can-cia.org/can/ttcan/
(3) OSEK/VDX ; OSEK/VDX Time-Triggered Operating System Version 1.0, July 2001.
(4) OSEK/VDX ; OSEK/VDX Operating System Version 2.2.1, Jan 2003.
(5) OSEK/VDX ; OSEK/VDX Communication Version 3.0.3, July 2004.
(6) OSEK/VDX ; OSEK/VDX Fault Tolerant Communication Version 1.0, July 2001.
(7) OSEK/VDX ; OIL: OSEK Implementation Language Version 2.5, July 2004.
(8)* Freescale Semiconductor社のMC9S12XFR128に関するWebサイト，http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MC9S12XFR128&nodeId=0162468636bJwn&tid=tFRlp

はっとり・ひろゆき
(株)ヴィッツ

<筆者プロフィール>
服部博行．自分の興味のあるものは自分で作りたがるやっかいな性格．日曜大工，ビール作りなどで自分の時間を楽しんでいる．最近は，2人乗りの車でドライブを楽しみ，家族からは白い目で見られているが，本人はまったく気にしていない．